1. **Insert a Node**: Write a function to insert a node into a BST.
2. **Delete a Node**: Implement deletion of a node from a BST.
3. **Search a Node**: Write a function to search for a value in a BST.
4. **Find Minimum Value**: Implement a function to find the minimum value in a BST.
5. **Find Maximum Value**: Implement a function to find the maximum value in a BST.
6. **Inorder Traversal**: Write a function to perform an inorder traversal of a BST.
7. **Preorder Traversal**: Write a function to perform a preorder traversal of a BST.
8. **Postorder Traversal**: Write a function to perform a postorder traversal of a BST.
9. **Level Order Traversal**: Implement level order traversal using a queue.
10. **Count Nodes**: Write a function to count the total number of nodes in a BST.

**Advanced Operations**

1. **Check if a Tree is a BST**: Write a function to determine if a given tree is a BST.
2. **Validate BST**: Implement a function to validate whether a tree is a valid BST.
3. **Find Height of a BST**: Write a function to find the height of a BST.
4. **Check for Balanced BST**: Write a function to check if a BST is height-balanced.
5. **Lowest Common Ancestor (LCA)**: Implement a function to find the LCA of two nodes in a BST.
6. **Convert Sorted Array to BST**: Write a function to convert a sorted array into a balanced BST.
7. **Sorted List to BST**: Convert a sorted linked list into a height-balanced BST.
8. **Find kth Smallest Element**: Implement a function to find the k-th smallest element in a BST.
9. **Find k-th Largest Element**: Write a function to find the k-th largest element in a BST.
10. **Print Nodes at k Distance**: Implement a function to print all nodes at a distance k from a given node.

**Tree Transformations**

1. Flatten BST to Sorted Linked List: Convert a BST to a sorted linked list.
2. Mirror of BST: Write a function to create a mirror image of a BST.
3. Merge Two BSTs: Merge two BSTs into a single BST.
4. **Find Successor and Predecessor**: Write a function to find the successor and predecessor of a given node.

Nodes: 30, 20, 40, 10, 25, 35, 50

Task: Insert the nodes into a BST in the specified order. After each insertion, check the balance factor and perform any necessary AVL rotations. Illustrate the final structure of the tree.

Question 3

Nodes: 50, 30, 70, 20, 40, 60, 80, 10

Task: Insert the provided nodes into a BST, performing AVL rotations as necessary after each insertion to maintain balance. Show the tree after each insertion, including any rotations that occur.

Question 4

Nodes: 40, 30, 20, 10, 25, 35, 50, 60

Task: Construct a BST using the given nodes. After each insertion, display the tree structure and any AVL rotations needed to keep the tree balanced.

### Steps to Construct BST from Inorder and Postorder

1. **Understand the Traversals**:
   * **Inorder Traversal**: Left subtree → Root → Right subtree
   * **Postorder Traversal**: Left subtree → Right subtree → Root
2. **Identify the Root**:
   * The last element of the postorder traversal is the root of the BST.
3. **Find the Root in Inorder**:
   * Locate this root element in the inorder traversal. This divides the inorder array into two parts:
     + Left subtree elements (to the left of the root in inorder)
     + Right subtree elements (to the right of the root in inorder)
4. **Recursive Construction**:
   * Recursively apply the same process to construct the left and right subtrees:
     + For the left subtree:
       - Use the elements to the left of the root in the inorder array and the corresponding elements from the postorder array.
     + For the right subtree:
       - Use the elements to the right of the root in the inorder array and the corresponding elements from the postorder array.
5. **Base Case**:
   * If there are no elements left in the inorder or postorder arrays, return

Let's say we have the following traversals:

* **Inorder**: [D, B, E, A, F, C]
* **Postorder**: [D, E, B, F, C, A]

Draw the tree